Fast Search Algorithm for Sequences of Hierarchically Structured Data

Masaki Murata[1], Masao Utiyama[1], Toshiyuki Kanamaru[1,2], and Hitoshi Isahara[1]

[1] National Institute of Information and Communications Technology,
3-5 Hikaridai, Seika-cho, Soraku-gun, Kyoto, 619-0289, Japan,
{murata,mutiyama,isahara}@nict.go.jp,
WWW home page: http://www.nict.go.jp/jt/a132/members/murata/
[2] Kyoto University,
Yoshida-nihonmatsu-cho, Sakyo-ku, Kyoto, 606-8501, Japan,
kanamaru@hi.h.kyoto-u.ac.jp

Abstract. We developed an algorithm for quickly searching sequences of hierarchically structured data, such as tagged corpora where each word includes information on its part of speech (POS) and minor POS and the word itself. Using our method, we first make a data item where each data item in a lower level is surrounded by two data items in a higher level. We then connect these data items to make a long string and store the string in a database. We use suffix arrays to query the database. Our experiments showed that our method was 194 times faster than a conventional method at fastest and 24 times faster on average. Our method can be used for other kinds of hierarchically structured data, such as Web applications. Methods that can be used on such data are in high demand. For example, our method can be used to retrieve Web text that includes hierarchical information of low, middle, and high semantic levels. If we use our method for such Web text, we can query using the terms, "High semantic level: method", "Word: in", and "Low semantic level: group"; in other words, our retrieval method is more useful and convenient than conventional Web retrieval.

1 Introduction

We developed an algorithm for quickly searching sequences of hierarchically structured data.[3] For example, in the Kyoto Text Corpus [1], each word has information on its part of speech (POS) and minor POS and the word itself. (A minor POS is a more specific POS.) The POS, minor POS, and word can be considered data of the highest layer, the second-highest layer, and the lowest layer. Hierarchically structured data, as explained below, has data from the lowest to the highest layers. The Kyoto Text Corpus has such a structure, so it can be considered to be sequences of hierarchically structured data. The algorithm we propose is for quickly searching sequences of such data. Our algorithm

[3] We obtained a Japanese patent for the algorithm.

Table 1. Example of sequences of hierarchically structured data.

Word	Minor POS	POS
The	definite article	article
technology	common noun	noun
in	preposition	particle
Japan	proper noun	noun
is	copula	verb
...	...	...

can be used not only for the Kyoto Text Corpus, but also for any sequences of hierarchically structured data.

An application of our algorithm is to search sequences of hierarchically structured data, such as a tagged corpus where each word includes information on its POS and minor POS and the word itself, as mentioned above. Our algorithm is useful for quickly searching a natural language processing system that uses such a tagged corpus. Linguists or users (i.e., learners) that would like to use linguistic information would want to quickly search the database using grammatical information such as POSs as mentioned above. Our algorithm is also useful for this case. Our method can be used for other kinds of hierarchically structured data, such as Web applications. Methods that can be used on such data are in high demand. For example, our method can be used to retrieve Web text that includes hierarchical information of low, middle, and high semantic levels. If we use our method for such Web text, we can query using the terms, "Higher semantic level: method", "Word: in", and "Low semantic level: group"; in other words, our retrieval method is more useful and convenient than conventional Web retrieval.

2 Algorithm

2.1 How to store data in a database

In our algorithm, we use suffix arrays [2] for retrieval. We can perform fast searches by storing data in a database in a special form.

Data are stored in the following form. In our algorithm, we first create a data item where each data item in a lower level is surrounded by a pair of higher-level data items. We then connect these data items to make a long string and store the string in a database. We make indexes for fast searches by using suffix arrays.

For example, assuming that we were going to store the sentence in Table 1 in a database, we would first transform the word "The" into the appropriate form for the database. For this data item, the lowest level data is "The", and the second lowest data level is "definite article", so we put "definite article" on both sides of "The" and obtain the expression "definite article:The:definite article". We use ":" as a boundary between the data items. The third lowest data item is "article", so we put "article" on both sides of "definite article:The:definite article" and obtain the expression, "article:definite article:The:definite article:article". This is the complete expression for the data item "The" stored in the database. We transform the words "technology", "in", "Japan", and "is" in the same way and obtain the following string.

```
/article:definite article:The:definite article:article/
noun:common noun:technology:common noun:noun/
particle:preposition:in:preposition:particle/
noun:proper noun:Japan:proper noun:noun/
verb:copula:is:copula:verb/
...
```

We store this string in a database with a "/" between expressions. Above, the string is separated into words, and each expression is on a separate line for easy viewing. However, in the database, we do not separate the expressions. Instead, we connect them into strings.

2.2 Method of retrieval

When we query the database, we transform the query into a string as explained above because the database is constructed from hierarchically structured data. However, we transform data items at the beginning and end of a query in different ways. Data items at the beginning of a query are put in order from the lowest layer to highest. Data items at the end of a query are put in order from the highest layer to lowest.

For example, assume that we make the following query.

```
common noun, noun
in, preposition, particle
proper noun, noun
```

That is, the POS of the first word in the query is noun, and its minor POS is common noun; the second word is "in", its POS is particle, and its minor POS is preposition; the POS of the third word is noun, and its minor POS is proper noun.

In this case, the data item at the beginning of the query is transformed to "common noun:noun/" in order from the lowest layer to highest. The data item in the middle of the query is transformed to "particle:preposition:in:preposition: particle/", where each data item in a lower level is surrounded by a pair of higher-level data items. The data item at the end of the query is transformed to "noun:proper noun" in order from the highest layer to lowest. As a result, we construct the following string from the query.

```
common noun:noun/
particle:preposition:in:preposition:particle/
noun:proper noun
```

We search for this string in a database by using a suffix array.

Suffix arrays are known broadly as algorithms for quickly searching for a substring in a string. The algorithm has $log(n)$ steps when the length of the string is n.

Our algorithm can search for a complicated hierarchically structured datum by searching only once using a suffix array.

Although we showed the case where the query is a sequence of three data items (words) and three layers are used for a hierarchical structure, our method can handle sequences of more than three items and more than three layers.

Now, we assume that we query the following.

common noun, noun
particle

That is, the first word's POS is noun, its minor POS is common noun, and the second word's POS is particle. In this case, the data item for the first word is transformed into "common noun:noun" in order from the lowest layer to highest. The data item for the second word is transformed into "particle" in order from the highest layer to lowest. The transformed query is as follows:

common noun:noun/particle

In this case, too, our algorithm can search for a hierarchically structured datum by searching only once using a suffix array.

2.3 Supplement

In our algorithm, higher-level data items are put on both sides of lower-level data items. If we adopt another method where higher-level data items are put on one side of lower-level data items, our algorithm cannot be used to search. For example, when we adopt the wrong method, the data in Table 1 is stored as follows.

/The:definite article:article/
technology:common noun:noun/
in:preposition:particle/
Japan:proper noun:noun/
is:copula:verb/
...

In this case, the query that the first word's POS is noun, its minor POS is common noun, and the second word's POS is particle as in the following cannot be retrieved using a single search with a suffix array.

common noun:noun
particle

This is because an obstructive string, "in:preposition:", is between "common noun:noun" and "particle", and a query cannot be expressed with an unsplit string. Instead, in our algorithm, the data sequence is stored as follows.

/article:definite article:The:definite article:article/
noun:common noun:technology:common noun:noun/
particle:preposition:in:preposition:particle/
noun:proper noun:Japan:proper noun:noun/
verb:copula:is:copula:verb/
...

In this case, what separates "common noun:noun" and "particle" is a separation symbol, "/", only. Therefore, when we would like to search the following query, we make the transformed query, "common noun:noun/particle".

common noun:noun
particle

We can make the query by searching only once using a suffix array.

2.4 Notice for the algorithm

In our algorithm, a data sequence that can be searched for using a one-time retrieval must fulfill the following conditions: The data items in the middle of the query must have data for all the layers, from the lowest to highest, and the data items at the beginning or end of the query must have data for all the layers that are higher than a certain layer. A data sequence must consist of consecutive items, so in a data sequence, there must not be gaps. A data sequence that does not satisfy the above conditions cannot be expressed with a connective string and cannot be searched for using one-time retrieval.

For example, the query specifying that the first word's minor POS is common noun and the second word's POS is particle, as follows, cannot be searched for using one-time retrieval.

common noun
particle

A string in the database is as follows.

noun:common noun:technology:common noun:noun/
particle:preposition:in:preposition:particle/

The word "noun" is between "common noun" and "particle". Therefore, the query cannot be expressed with a connective string and cannot be searched for using one-time retrieval.

To solve the problem, we should use our knowledge of hierarchically structured data and complement data items in higher layers before making the query. For example, the data item in the layer above "common noun" is "noun". Therefore, when the query is the following,

common noun
particle,

Table 2. Average retrieval time.

Retrieval time in conventional method	Retrieval time in our method	Ratio of retrieval time
0.410 s	0.017 s	24.3

"noun" is the complement of "common noun"[4], so we assume that the query is the following.

common noun, noun
particle

The transformed query is "common noun:noun/particle", and we can make the query by using a one-time retrieval.

3 Experiments

Our algorithm is very fast because it uses suffix arrays, a fast search algorithm, only once. We experimented to determine by how much our algorithm is faster than a conventional method. In the experiments, we used our algorithm and the following conventional method.

In the conventional method, we first divide a query into two parts. We next search for the first part of the query using a suffix array. We then check whether each result of the first search has the second part of the query and output the results having the second part of the query as the final retrieval results.

We used Sun UE420R (450 MHz) systems as the computers for the experiments. We used C to program them.

We used the Kyoto Text Corpus Version 2.0 [1] (which has about 20,000 Japanese tagged sentences) as a database. We used the POS information as the data in the highest layer, the minor POS information as the data in the middle layer, and the word information as the data in the lowest layer. For example, in the data item "technology, common noun, noun", "noun" is treated as the data item in the highest layer, "common noun" is treated as the data item in the middle layer, and "technology" is treated as the data item in the lowest layer.

The experimental results are shown in Tables 2 to 4. In the experiments, we used the 24 expressions shown in the first column of Table 3 as the first parts of our queries and used the 51 expressions shown in the first column of Table 4 as the second parts. We made 1224 (= 24 × 51) queries by making all the combinations of the first and second parts and used the queries for the experiments. In our method, we connect the first and second parts of a query

[4] In some cases, a data item in a lower layer has plural data items in a higher layer. In such a case, we have to make queries using all the data items and make OR retrieval using the queries, have the user select one of the data items and make the retrieval using the query made using it, or use word sense disambiguation techniques to select an appropriate data item and make the retrieval using a query made using it.

Table 3. Retrieval time for first part of query.

First part of query	Number of items output	Time (s) (conventional)	Time (s) (our method)	Ratio of time
verb	48692	1.123	0.017	65.1
adjective	11213	0.274	0.018	15.5
prefix	3150	0.082	0.017	4.7
adnoun	532	0.025	0.016	1.5
pronoun	4513	0.120	0.017	7.1
noun	178487	3.955	0.020	194.0
verbal noun:noun	45110	1.103	0.019	58.6
common noun:noun	87130	2.085	0.017	120.8
formal noun:noun	4704	0.130	0.015	8.5
person name:noun	6845	0.172	0.015	11.7
location name:noun	10196	0.257	0.016	16.0
organization name:noun	3249	0.091	0.019	4.9
proper noun:noun	97	0.020	0.018	1.1
prime minister:common noun:noun	447	0.030	0.018	1.7
goverment:common noun:noun	529	0.033	0.019	1.7
expectation:verbal noun:noun	152	0.018	0.016	1.1
hito (human):common noun:noun	542	0.030	0.015	2.0
Murayama:person name:noun	232	0.021	0.016	1.3
Tokyo:location name:noun	390	0.023	0.014	1.7
mono (thing):formal noun:noun	681	0.035	0.017	2.0
koto (matter):formal noun:noun	2255	0.088	0.019	4.7
no (thing):formal noun: noun	1350	0.061	0.015	4.1
omou (think)::verb	120	0.020	0.015	1.4
aru (exist)::verb	1228	0.054	0.018	3.0

into one string and search for it in a database. Using the conventional method, we first search for the first part of a query in a database and check whether each result of the first search has the second part of the query. We then output the results having the second part of the query as the final retrieval results.

Table 2 shows the average retrieval times for the conventional method and our method and the ratio of the average retrieval time of the conventional method to that of our method. Our method is very fast (0.017 s), and the conventional method is slower (0.410 s). Our method is 24 times faster than the conventional method. This indicates that our method is effective.

Table 3 shows the average retrieval time for the queries where the first part of the query is the corresponding first part in the first column of the table and the second part is any of the second parts from Table 4. Table 4 shows the average retrieval time for the queries where the second part of the query is the corresponding second part in the first column of the table and the first part is any of the first parts from Table 3. In the tables, "Number of items output" indicates the number of items output using only the first or second part of the query. (That is, "Number of items output" indicates the number of expressions matching the first or second part of the query.) "Time (conventional)" indicates the average retrieval time for the conventional method. "Time (our method)" indicates the average retrieval time for our method. "Ratio of time" indicates

Table 4. Retrieval time for second part of query.

Second part of query	Number of items output	Time (s) (conventional)	Time (s) (our method)	Ratio of time
verb	48692	0.396	0.015	27.2
aux. verb	4074	0.417	0.018	23.3
adjective	11213	0.406	0.015	26.4
postfix	37322	0.420	0.020	20.6
copula	4616	0.403	0.018	22.5
noun	178487	0.408	0.020	20.4
noun:verbal noun	45110	0.400	0.016	24.6
noun:common noun	87130	0.425	0.018	24.3
noun:formal noun	4704	0.407	0.015	27.1
noun:person name	6845	0.425	0.014	30.9
noun:location noun	10196	0.405	0.017	23.7
noun:organization noun	3249	0.420	0.017	25.2
noun:proper noun	97	0.392	0.018	21.4
particle	125877	0.417	0.022	18.5
particle:case particle	68951	0.410	0.018	22.3
particle:postfix particle	860	0.422	0.015	27.4
particle:sub particle	24565	0.394	0.019	21.0
particle:connective particle	31501	0.410	0.020	20.9
particle:case particle:*kara* (from)	2286	0.419	0.017	24.5
particle:case particle:*ga* (sub.)	12240	0.404	0.014	28.5
particle:case particle:*de* (in)	6901	0.400	0.014	29.1
particle:case particle:*to* (with)	11248	0.402	0.021	18.9
particle:case particle:*ni* (to)	15697	0.416	0.018	23.8
particle:case particle:*no* (sub.)	1601	0.421	0.018	24.1
particle:case particle:*ha* (sub.)	1	0.394	0.015	25.5
particle:case particle:*he* (to)	883	0.426	0.018	23.3
particle:case particle:*made* (to)	795	0.395	0.016	24.9
particle:case particle:*yori* (from)	232	0.426	0.020	20.9
particle:case particle:*wo* (obj.)	17064	0.411	0.016	25.3
particle:postfix particle:*ka*	699	0.407	0.013	31.5
particle:postfix particle:*ne*	47	0.405	0.019	21.6
particle:postfix particle:*yo*	41	0.406	0.019	21.2
particle:postfix particle:*wa*	4	0.407	0.017	24.4
particle:sub particle:*kurai* (about)	19	0.412	0.014	29.9
particle:sub particle:*gurai* (about)	17	0.422	0.022	19.5
particle:sub particle:*koso* (even)	87	0.417	0.016	26.3
particle:sub particle:*sae* (even)	52	0.403	0.015	27.7
particle:sub particle:*shika* (only)	127	0.404	0.015	26.9
particle:sub particle:*sura* (even)	41	0.404	0.015	27.7
particle:sub particle:*dake* (only)	580	0.415	0.016	25.5
particle:sub particle:*datte* (even)	11	0.417	0.016	25.7
particle:sub particle:*demo* (even)	482	0.407	0.017	24.4
particle:sub particle:*nado* (etc.)	1555	0.417	0.018	23.3
particle:sub particle:*nomi* (only)	24	0.412	0.018	23.5
particle:sub particle:*bakari* (only)	49	0.408	0.016	25.1
particle:sub particle:*made* (also)	6	0.408	0.015	26.5
particle:sub particle:*mo* (also)	4663	0.422	0.016	26.6
particle:connective particle:*to* (and)	15	0.417	0.013	32.3
particle:connective particle:*no* (of)	25739	0.400	0.015	25.9
particle:connective particle:*ya* (or)	1530	0.409	0.019	21.8
particle:connective particle:*mo* (too)	7	0.419	0.017	24.5

the ratio of the average retrieval time of the conventional method to that of our method.

From Table 3, we found that when the number of items output using the first part of a query is larger, the time ratio of the conventional method to that of our method is also larger. That is, when the number of items output using the first part of a query is large, the conventional method needs much more time than our method for retrieval. For example, when the first part of a query is "noun", the number of items output is very large (178487). Therefore, the conventional method needs 4 s. On the other hand, our method is very fast (0.020 s). The ratio of time is 194, so our method is 194 times faster than the conventional method. This indicates that our method is effective.

On the other hand, when the number of items output using the first part of a query is small, the ratio of time is not so large. For example, when the first part of a query is "proper noun:noun", the number of items output is small (97). In this case, the retrieval times of the conventional method and our method are 0.020 and 0.018 s and are very similar.

Next, let us examine the results for the second parts of the queries using Table 4. In this case, the conventional method needs almost the same time for each query. We found that the retrieval time of the conventional method does not depend on the second part of a query.

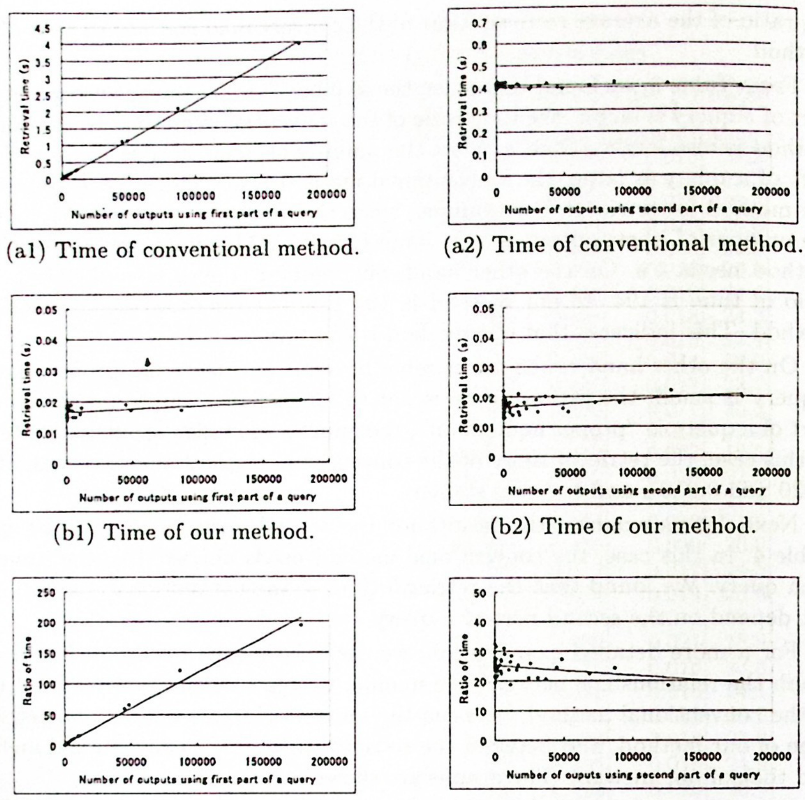
For a more detailed examination, we used the results in Tables 3 and 4 to graph the relationships between the number of items output and retrieval time of the conventional method, between the number of items output and retrieval time of our method, and between the retrieval time of the conventional method and that of our method. The graphs are shown in Figure 1.

Using Figure 1(a1), we found that the retrieval time of the conventional method is roughly proportional to the number of items output using the first part of a query. The reason is that the conventional method has to check whether each result in the first search has the second part of the query.

Next, using Figure 1(a2), we can see that the retrieval time of the conventional method does not depend on the number of items output using the second part of a query. The reason is that the conventional method has to check whether each result in the first search has the second part of the query, and the retrieval time does not depend on the number of data items in a database corresponding to the second part of the query.

Using Figures 1(b1) and 1(b2), we can see that the retrieval time of our method is slightly longer when the number of items output using the first or second part of a query is larger.

Next, we examined the ratio of the retrieval time of the conventional method to that of our method using Figures 1(c1) and 1(c2). Because the ratio is the retrieval time of the conventional method divided by that of our method, the values in Figures 1(a1) and 1(a2) divided by the values in Figures 1(b1) and 1(b2) are the values in Figures 1(c1) and 1(c2). Because the values in Figure 1(a1), which form a straight line from the lower left side to the upper right side, are divided by the values in Figure 1(b1), which form a straight line tilted down



(a1) Time of conventional method. (a2) Time of conventional method.

(b1) Time of our method. (b2) Time of our method.

(c1) Ratio of time. (c2) Ratio of time.

Fig. 1. Retrieval times for first or second part of query.

slightly on the left, the values in Figure 1(c1) go up to the right but curve down slightly compared to a straight line. Because the values in Figure 1(a2), which form a straight line from left to right, are divided by the values in Figure 1(b2), which form a line going down slightly to the left, the values in Figure 1(c2) go down slightly to the right.

From these results, we found that our method is especially effective when the number of items output using the first part of a retrieval is large. We also found that when the number of items output using the second part of a retrieval is small, the speed of our method is slightly reduced, and the difference between the speeds of our method and the conventional method is slightly smaller.

In the conventional method described above, we always used the first part of a query first and checked whether each result of the first search had the second part of the query to more easily analyze experimental results. However, in general, we first determine whether the first or second part of a query has the smaller

Table 5. Average retrieval time

Retrieval time of conventional method 2	Retrieval time of our method	Ratio of retrieval time
0.112 s	0.017 s	6.65

Table 6. Example of data sequence that includes semantic information.

Word	Semantic Level 2	Semantic Level 1
The	None	None
technology	Method	Abstraction
in	None	None
Japan	Country	Group
is	None	None
...	...	...

number of items to output, then use the part with the smaller number for the first retrieval, and check whether each result in the retrieval has the other part of the query. This is because when the number of the outputs in the first retrieval is smaller, the conventional method is faster. Therefore, we experimented using this method. We call it *conventional method 2*. The results are shown in Table 5. The retrieval time of conventional method 2 is 0.112 s, which is faster than the conventional method (0.410 s). However, the retrieval time of conventional method 2 is 6.2 times slower than that of our method.

We can use SQL to handle hierarchically structured data, so we experimented using it. However, when we used MySQL, we needed a few seconds to search for only one word[5], and we needed a few minutes to a few hours to search for a combination of two words[6]. Therefore, our method is much faster at retrieving one word (100 times faster than the method using SQL; our method needs 0.02 s, and the method using SQL needs a few seconds) and even faster at retrieving a pair of words (6000 to 360000 times faster than the method using SQL; our method needs 0.02 s, and the method using SQL needs a few minutes or a few hours) than the method using SQL.

4 Application

Our algorithm can be used not only for the Kyoto Text Corpus, but also for any sequences of hierarchically structured data.

For example, our method can be used for data sequences using the semantic information of a hierarchical thesaurus such as in Table 6. In this case, the data sequence in the table is transformed into the following string to store it in a database.

[5] The retrieval time for "noun" was 3.59 s.

[6] The retrieval time of the pair "noun:common noun" and "*de* (in):particle" was 3.5 hours.

/None:None:The:None:None/
Abstraction:Method:technology:Method:Abstraction/
None:None:in:None:None/
Group:Country:Japan:Country:Group/
None:None:is:None:None/
...

In the data, we can retrieve a query using the terms, "Semantic level 2: Method", "Word: in", and "Semantic level 1: Group"; that is, we can retrieve the semantic query of "Method in Group". In this case, we complement the query with some data items using our knowledge of hierarchical thesauruses such as WordNet [3] and make the string "Method: Abstraction/None:None:in:None:None/Group". By using this string, we can make the query using a one-time retrieval.

5 Conclusion

We developed an algorithm for quickly searching sequences of hierarchically structured data, such as a tagged corpus where each word includes information on its part of speech (POS) and minor POS and the word itself. To determine the effectiveness of our method, we experimented to compare the retrieval time of our method with that of a conventional method. The conventional method was the method dividing the query into two parts, making the first retrieval using one of the two parts, and checking whether each of the results in the first retrieval included the other part or not. The experimental results showed that our method was 194 times faster than the conventional method at fastest and 24 times faster on average. We also found that our method is especially effective when the number of items output using the first retrieval is large.

Our algorithm can be used for other kinds of sequences of hierarchically structured data. We showed sequences of hierarchically structured data where each word has a higher semantic label and a lower semantic label and described how to transform a sequence of such data into a string to store the data in a database.

Our technique for putting semantic information into text can be used for Web text. Our method can therefore be used for Web retrieval, which is in high demand. If we use our method for Web text, we can query using the terms, "Semantic level 2: Method", "Word: in", and "Semantic level 1: Group"; in other words, our retrieval method is more useful and convenient than conventional Web retrieval.

References

1. Sadao Kurohashi. Kyoto Text Corpus Version 2.0. 1998. (in Japanese).
2. U. Manber and G. Myers. Suffix Arrays: a newmethod for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.
3. Princeton University. WordNet 2.0. 2003.